Customizable Knowledge Graph Visualization using the Whyis Knowledge Explorer

Jamie P. McCusker[1,*,†]

[1]Rensselaer Polytechnic Institute, Troy, NY, USA

Abstract

Network visualization over large knowledge graphs suffers from multiple challenges: graphs have varying and sometimes multiple ways to represent what people expect a "link" to be - everything from direct triples to complex chemical interactions, social constructs, and OWL property restrictions can be considered a link. Additionally, large knowledge graphs cannot be usefully visualized as a whole because they are simply too large and complex, an any patterns are lost in the noise when there is enough computational ability to represent them. The Whyis Knowledge Explorer is a component of the Whyis knowledge graph development framework that addresses these issues. It allows for fast, customizable network visualization of large scale knowledge graphs. By providing a "starting point" with any specific node, users can explore the graph piece by piece, building a view up by expanding selected nodes on demand, making it easier to explore locally. By using "data views", the component provides a consistent user interface over a wide range of entity types that can handle both simple and complex relationships between entities. These data views publish a consistent output from multiple templates and can be extended through plugins as well as by the implementing Knowledge Graph App (KGApp). Entity types can also be assigned custom styles through CSS using Cytoscape.js styling. Additionally, links can be qualified with certainty values, showing more probable links as having greater weight. We also use the same interface to provide a summary view of the knowledge graph by automatically generating concept maps of instantiated types, allowing users to see and explore overall usage patterns in the knowledge graph, highlighting both intended design and knowledge curation issues. This component has been a key part of many Whyis-based projects and is mature and scalable.

Keywords

visualizations, web applications, knowledge graphs

1. Introduction

Current visualization strategies for knowledge graphs (KGs) either do not take into account the graph design decisions for things like reified links or spurious links that are not relevant to the task, or they provide a fully customized visualization that is difficult to repurpose for other projects. In a similar vein, most overall views of knowledge graph design are usually created by hand based on the knowledge of the KG developer and may not always reflect the state of the current graph.

VOILA 2024: The 9th International Workshop on the Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs co-located with the 23rd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 11-15, 2024.

*Corresponding author.
mccusj2@rpi.edu (J. P. McCusker)
https://tw.rpi.edu (J. P. McCusker)
0000-0003-1085-6059 (J. P. McCusker)

We have built knowledge explorer network visualization tool in Whyis that lets developers customize the way links from specific entities are rendered. Whyis enables software developers to create knowledge graph applications (KGApps) using minimal modifications, allowing for the use of code-oriented deployment and management tools like GitHub, Docker, and other DevOps tools. This low code/no code approach allows creators of knowledge graphs to do so without coding, but if they need to add code, it would be minimal. In a previous paper [1] we illustrated the capabilities and structure of Whyis using a demonstration knowledge graph of characters and their interactions from the novel *Les Miserables*, as originally created by Donald Knuth and maintained by Media and Design Studio[1] We extended this demo to highlight the network visualization capabilities included in Whyis, and is available on GitHub[2].

1.1. Whyis UI and Data Views

A key feature of Whyis for visualization and knowledge interaction is the use of type-driven custom views for nodes by both the *rdf:type* of the node and the *view* URL parameter, which drives all user interface and API interactions within Whyis. Every URL in Whyis is a node in the knowledge graph, the URI of which can be linked data if the configured LOD prefix matches the server URL prefix. Additionally, other nodes can be viewed by using the URI as a URL parameter. And additional URL parameter, `view`, can be added to provide alternate view of the entity, which can be configured like this:

```
:myCustomView rdfs:subClassOf whyis:hasView;
    dcterms:identifier "custom".
```

This creates a new view type called `custom` that can then be rendered. Entity types can be registered with specific template to render particular views using a *type view template* triple. For instance, below we register a default view for foaf:Person using the `person_view.html` template and a `custom` view using the `person_custom.json` template:

```
<http://xmlns.com/foaf/0.1/Person> a owl:Class;
    whyis:hasView "person_view.html";
    :myCustomView "person_custom.json".
```

These templates are rendered using Jinja2 [3] and can access a large collection of APIs to access the current entity, the knowledge graph as a whole, and other functions.

1.2. Whyis Data Views are Open Contracts

Design by Contract [2] is a defensive programming strategy that encourages clearly defined interfaces between software modules, so that users of a module know what to expect from it, and implementers of the module know what they are expected to provide. By specifying a clear interface, it is possible to create multiple implementations that provide the same service, which become essentially open contracts. Data views in Whyis serve as a form of open contract. By

[1]Available at https://github.com/MADStudioNU/lesmiserables-character-network.
[2]https://github.com/whyiskg/les-mis-demo
[3]http://jinja.pocoo.org

reimplementing a named view on an entity type that is different from the reference implementation, the implementer is expected to provide a compatible data structure to the orignal. This allows for customization of the content served from a given view for a user interface, but it also allows other user interfaces to be created that take advantage of that same view. UI views and data views can therefore be matrixed together in ways that would otherwise be impossible. For instance, a *label* view can be used by multiple UIs (and even other data views) as-is, because it can be relied on to provide a human readable text string ready to be embedded in other content, while what counts as a label might vary from entity to entity based on its type.

This paper provides the following contributions. We developed a "design by contract"-based API using Whyis typed views for node and link traversal in a knowledge graph can improve network visualization in those graphs. This approach can also support multiple ways of rendering nodes and links. Finally, we also show how graph summarization algorithms can use the same framework for visualization and exploration.

2. Approach

In order to implement a design by contract approach, we provide a general-purpose user interface in Javascript that iteratively requests up to two views from a given entity:

outgoing Return all links for which the entity in question is a subject, or source node, in the link.

incoming Return all links for which the entity in question is a subject, or target node, in the link.

The user interface uses Cytoscape.js [3] to render and layout the network visualizations. While the UI is available within the `explore` view, it is implemented as an AngularJS directive and can therefore be embedded in any other view in the knowledge graph.

The `incoming` and `outgoing` views for every resource in the graph are required to implement the following JSON structure:

```
[
    {
        "articles": ["[URI]", ...] ,
        "from": ["[URI]", ...] ,
        "link": "[URI]",
        "link_types": [
            {
                "label": "[string]",
                "uri": "[URI]"
            }
        ],
        "probability": [float: 0-1],
        "source": "[URI]",
        "source_label": "[string]",
        "source_types": [ "URI", ...],
        "target": "URI",
        "target_label": "string",
        "target_types": [ "URI", ...]
    },
    ...
]
```

The approach used here tries to keep as much of the JSON flattened to make it easier to use as much as possible via a SPARQL query. However, this can be implemented in many different ways, including passing off the entire JSON rendering to code outside of the template. For instance, the default view uses a template filter called `probit` to provide most of the query, while needing to provide a query fragment. For instance, this template can be used for generic resources:

```
{{'''graph ?assertion {
      {
         ?source ?link_type ?target.
      }
    }
    bind(?assertion as ?link)
    minus { ?source sio:hasPart|sio:hasAttribute ?target }
    minus { ?target a sio:Term.}
    filter (!sameTerm(?source, ?target) && isIRI(?target))
''' | probit(source=this.identifier) | tojson | safe}}
```

The `probit` filter includes that query fragment in a larger query and searches for source and target types and labels. Within the Les Miserables knowledge graph, we use the following template, which takes advantage of the *schema:InteractAction* instances that we use in the graph (see Figure 1):

```
{{'''graph ?assertion {
      {
         ?article schema:participant ?source.
         ?article schema:participant ?target.
         ?article a ?link_type.
      }
    }
    bind(?article as ?link)
    bind(0.6 as ?probability)
    filter (!sameTerm(?source, ?target) && isIRI(?target))
''' | probit(source=this.identifier) | tojson }}
```

Conveniently, implementers can simply bind the source or the target depending on the view being rendered.

2.1. Automated Meta-Analysis

To provide a more unified view of links between entities, we distinguish between a specific link and the type of the link. For instance, each individual character interaction in the graph can be seen as evidence that any two characters know each other, but any one interaction is a weak indicator of that. Multiple interactions tend to accumulate that evidence quickly. In many scientific domains, evidence can be collected with varying levels of confidence in the evidence. Independent experiments that have the same finding (expressed as a link, interaction, type assertion, or other knowledge) need to be given a way to combine that knowledge in a meaningful way.

In Whyis, we provide a form of automated meta analysis using Stouffer's Z Method [4]. This method allows multiple experiments or observations with varying levels of confidence that produce the same results reinforce each other. For each independent observation of $P(A)$ in $P(A_1), P(A_2), ...P(A_k)$, we compute the $P(A)$ as follows, where ϕ is the standard normal cumulative distribution function and ϕ^{-1} is the inverse CDF:

$$P(A) = \phi\left(\frac{\sum_{i=1}^{k} \phi^{-}1\left(P\left(A_i\right)\right)}{\sqrt{k}}\right)$$

For example, let us say that there is an ensemble of to link predictors that provide certainty scores of $P = 0.75, 0.9$. The corresponding Z of each is $Z = 0.7, 1.3$, making the consensus $Z = 1.41$, and the corresponding $P = 0.92$, a small improvement over either prediction. Adding additional observations will provide proportionally better probabilities, allowing evidence to accumulate.

Whyis provides the `probit` filter (a special form of function) that can process basic graph patterns for multiple observations and perform this sort of automated meta-analysis. A base rate probability is configurable in Whyis for assertions that do not have associated probabilities so they can also provide a baseline level of evidence.

3. Evaluation

We demonstrate our visualization approach using the Les Miserables graph from [1], extending it to provide additional sub-types for most characters. Figure 1 shows the structure we use to express interactions between characters, where the links are reified rather than simple $(subject, predicate, object)$ triples. This reified interaction makes it easier to indicate that the interaction is undirected or bi-directional, but also offers the opportunity to record which chapter it occurred in, which is provided in the data. In other interaction-based knowledge graphs, like systems biology graphs [5], it allows models to provide additional provenance like locality, intensity, and how the interaction was determined, i.e. through prediction, measurement, simulation, or other means.

In Figure 2, the Knowledge Explorer is shown with all of the Stouffer-method aggregated interactions of the main characters. While completeness may be useful on a node by node basis, overall patterns are difficult to see with networks of this size. Because of this, the Knowledge Explorer is capable of filtering on the probability of each link. We showed in [5] that aggregate

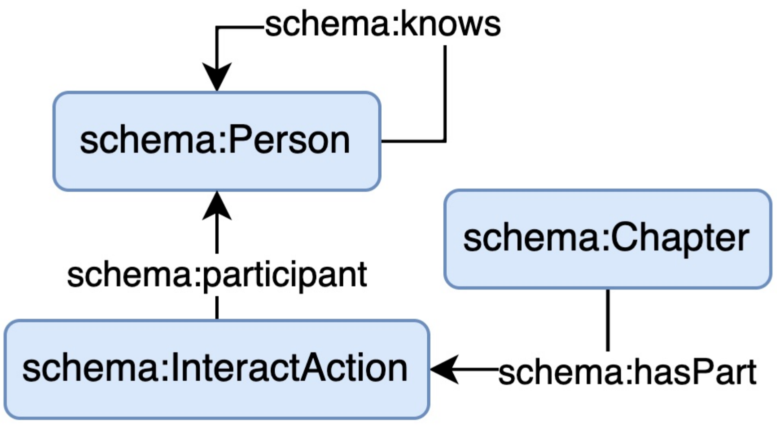

Figure 1: Basic structure of the Les Miserables knowledge graph, using Schema.org classes and properties.

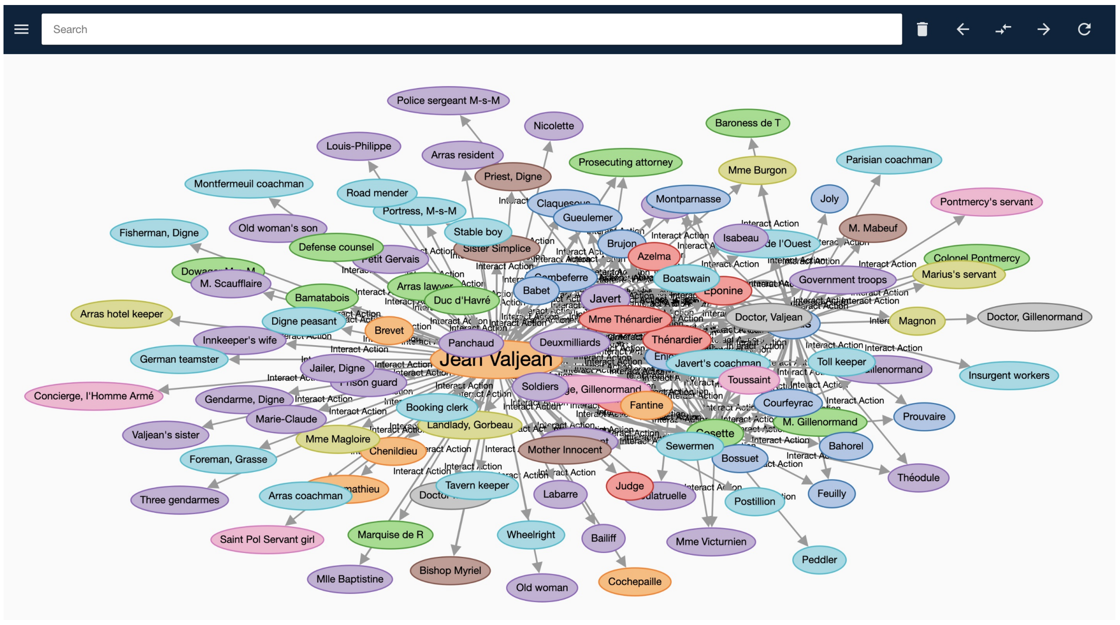

Figure 2: All links to characters that have directly interacted with the main characters of Les Miserables. To make graph browsing easier, Knowledge Explorer maps the relative size of displayed nodes to their centrality in the displayed graph.

probability across multiple links is a useful way to focus on high quality results. In Figure 3 we filter out links with $p >= 0.75$, resulting a useful graph of the main characters and the minor characters that they most frequently interact with. Note that in both cases, the nodes are color-coded by their type. The colors are auto-allocated from a 10 color palette, and adapt to the available types in the rendered graph.

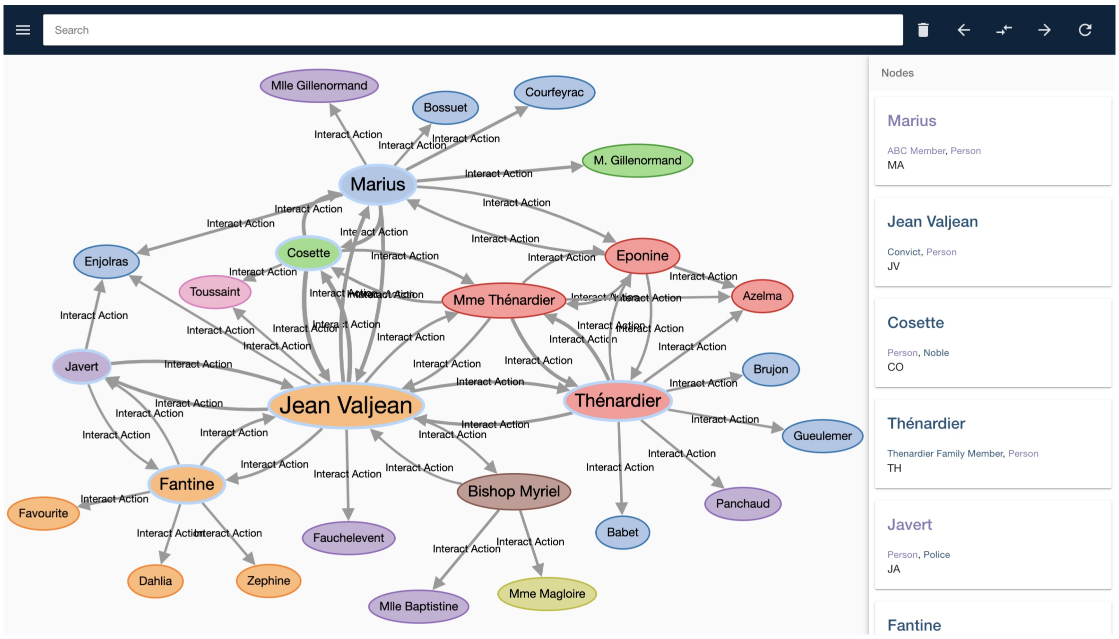

Figure 3: The network of characters that most frequently interacted with the main characters of Les Miserables, probability of > 0.75 using a base rate of 0.6 for each interaction.

In addition to the modeling used for character interactions, we created an agent that can summarize links between entities of different types to show how patterns are actually used in knowledge graphs. Called Concept Maps [6], they have been used for a long time to express graph patterns among different entity types in knowledge graphs. They have most commonly been used to plan out the structure of knowledge graphs. In this case, we invert the use case of concept maps and create them from the observed relationships between entities. We created a Whyis agent that creates a set of Relation objects in a nascent Knowledge Graph Modeling Ontology (KGMO). For each *owl:Class*, it looks at the frequency that its instances are connected to other instances of other classes via a given predicate. Future versions of this agent may be configured to analyze subclass relations for abstraction of these summaries, property paths, and other complex constructions. That frequency is recorded as a *sio:has-attribute* of the *kgmo:Relation* of type *kgmo:SyllogismConfidence*. Figure 4 provides more details of *kgmo:Relation* structures, including links to the subject type (*kgmo:hasSourceType*), predicate (*kgmo:property*), and object type (*kgmo:hasTargetType*). The resulting *kgmo:Relation* objects are then treated as links between classes in the knowledge graph (in addition to *rdfs:subClassOf* links), resulting in the summary visualization in Figure 4.

4. Related Work

Knowledge graph visualization using network tools has a long but varied history. Here we only consider tools that focus on network visualization in large scale query-able knowledge graphs, although there are graph visualization techniques from both Linked Data and ontologies [7].

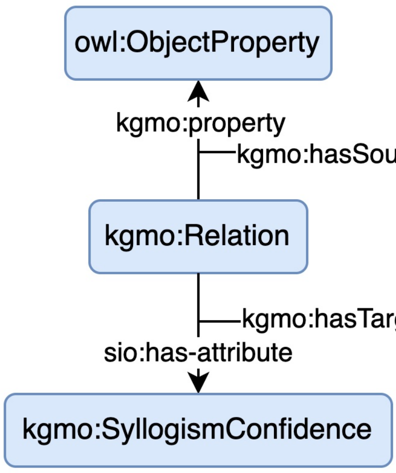

Figure 4: Structure of a KGMO relation.

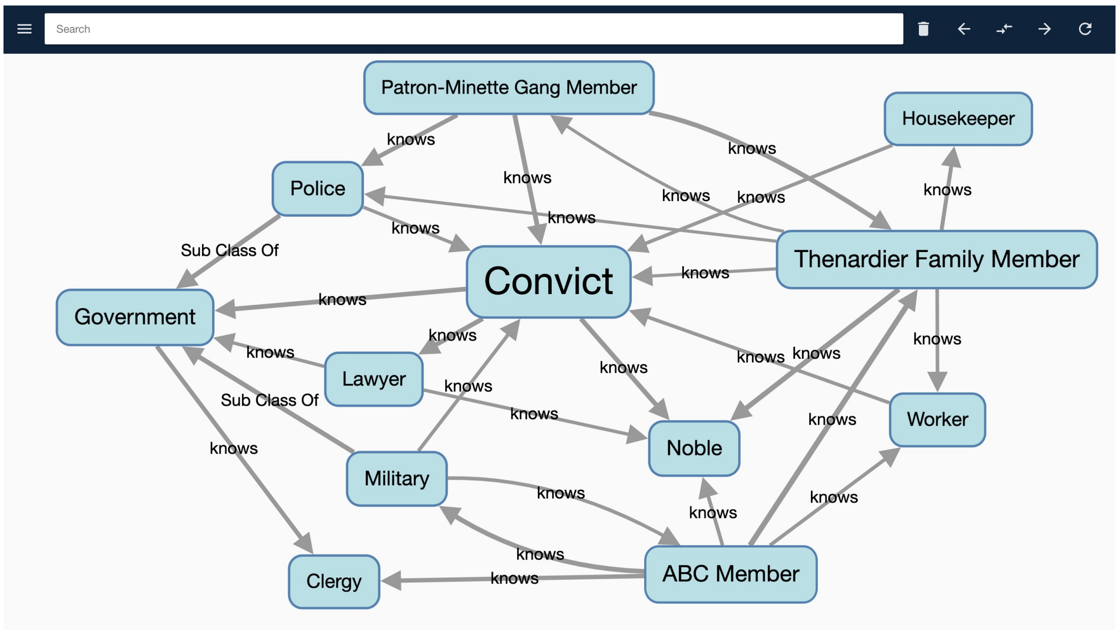

Figure 5: The types of characters that each one interacts with the most. For a link to be shown, at least half of all characters of the source type have interacted with a character of the target type.

Neo4J Browser allows developers to query the graph and visualize the results. However, it requires users to provide the query, limiting the potential audience [8]. Conversely, LODLive [9] provides a general purpose graph visualization tool, but it does not allow users to customize links based on their types. Since it is focused on traversing Linked Data, it also does not support aggregated knowledge graphs. ReDrugS [5] supports complex link types, but it was hard-coded to a specific domain. This is typical of most KG visualization approaches, and the Whyis approach was created to provide a generalized solution to this.

5. Discussion, Limitations, and Future Work

By implementing a Design by Contract API, the knowledge graph can enforce structured interactions with its data, ensuring consistency and reliability in how information is accessed and manipulated. This structured approach extends to visualization, where typed views for node and link traversal can enhance the clarity and usability of network representations. For example different stakeholders or applications can visualize the same graph in ways that suit their specific needs (e.g., emphasizing different types of nodes or relationships), although complex interactions would require development of new queries to support them (and therefore need expertise in SPARQL). Users can also define and utilize tailored views of the graph, optimizing visualization for their particular analytical tasks.

The use of graph summarization algorithms introduces another layer of utility. Algorithms can summarize large, complex graphs into more manageable forms without losing critical information, facilitating efficient exploration and analysis. Summarization results can also be visualized using the same framework as instance data, allowing users to grasp high-level insights quickly before delving into detailed exploration.

We plan to improve the Whyis Knowledge Explorer in a number of ways. First, we plan to incorporate the multi-hop searches for entity links that were available in ReDrugS [5]. This will be useful for systems biology analysis, but also any sort of analysis of multi-hop links between entities in general. Currently, the knowledge explorer is limited to a single, hard coded layout strategy. We hope to be able to let users select and configure layouts, and selectively apply them to nodes. This can allow for more interesting visual analyses. We also hope to explore more how shape constraints in both the knowledge graph using SHACL [10] and in data views using OpenAPI [11]. Finally we hope to provide more visual approaches on top of the existing `incoming` and `outgoing` data views, especially ones that hybridize network and quantitative visualization.

6. Conclusion

The Whyis Knowledge Explorer represents an advancement in the visualization and exploration of large-scale knowledge graphs. By addressing the challenges associated with graph representation and navigation, this tool offers a robust framework for developers and researchers alike. The ability to customize link rendering and employ type-driven views enhances usability and flexibility, accommodating both simple and complex relationships within the graph. Through its integration of design by contract API and sophisticated graph summarization algorithms, Whyis facilitates not only detailed node and link traversal but also insightful high-level analysis. This approach not only enhances the clarity of graph visualizations but also supports informed decision-making by stakeholders across various domains.

6.1. Availability

The Whyis Knowledge Explorer is a component of the Whyis knowledge graph development framework and is available through the Python Package Index and Dockerhub.

Python Package Index: whyis

License: Apache 2.0 License

Documentation URL: https://whyis.readthedocs.io

Docker pull command: docker pull tetherlessworld/whyis

Source Code URL: https://github.com/tetherless-world/whyis

Example Project URL: https://github.com/whyiskg/les-mis-demo

Acknowledgments

This work was funded by National Science Foundation (NSF) Award DMR-1310292, the Defense Advanced Research Projects Agency (DARPA) under Program #HR001122S0052 (ECOLE), the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the HIATUS Program #2022-22072200002, the National Institute of Environmental Health Sciences (NIEHS) Award 0255-0236-4609 / 1U2CES026555-01, IBM Research AI through the AI Horizons Network, the National Spectrum Consortium (NSC) project number NSC-17-7030, and by the Gates Foundation through Healthy Birth, Growth, and Development knowledge integration (HBGDki). Any opinions, findings and conclusions or recommendations expressed in this material are those the authors and do not necessarily reflect the views of AFRL, IBM, the Gates Foundation, ODNI, IARPA, DARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

[1] J. McCusker, D. L. McGuinness, Whyis 2: An open source framework for knowledge graph development and research, in: European Semantic Web Conference, Springer, 2023, pp. 538–554.
[2] B. Meyer, Applying'design by contract', Computer 25 (1992) 40–51.
[3] M. Franz, C. T. Lopes, D. Fong, M. Kucera, M. Cheung, M. C. Siper, G. Huck, Y. Dong, O. Sumer, G. D. Bader, Cytoscape. js 2023 update: a graph theory library for visualization and analysis, Bioinformatics 39 (2023) btad031.
[4] S. C. Kim, S. J. Lee, W. J. Lee, Y. N. Yum, J. H. Kim, S. Sohn, J. H. Park, J. Lee, J. Lim, S. W. Kwon, Stouffer's test in a large scale simultaneous hypothesis testing, Plos one 8 (2013) e63290.

[5] J. P. McCusker, M. Dumontier, R. Yan, S. He, J. S. Dordick, D. L. McGuinness, Finding melanoma drugs through a probabilistic knowledge graph, PeerJ Computer Science 3 (2017) e106. URL: https://doi.org/10.7717/peerj-cs.106. doi:10.7717/peerj-cs.106.

[6] P. Hayes, T. C. Eskridge, M. Mehrotra, D. Bobrovnikoff, T. Reichherzer, R. Saavedra, Coe: Tools for collaborative ontology development and reuse, in: Knowledge Capture Conference (K-CAP), volume 2005, 2005.

[7] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, E. Giannopoulou, Ontology visualization methods—a survey, ACM Computing Surveys (CSUR) 39 (2007) 10–es.

[8] J. Baton, R. Van Bruggen, Learning Neo4j 3. x: Effective data modeling, performance tuning and data visualization techniques in Neo4j, Packt Publishing Ltd, 2017.

[9] D. V. Camarda, S. Mazzini, A. Antonuccio, Lodlive, exploring the web of data, in: Proceedings of the 8th International Conference on Semantic Systems, 2012, pp. 197–200.

[10] J. Corman, J. L. Reutter, O. Savković, Semantics and validation of recursive shacl, in: The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17, Springer, 2018, pp. 318–336.

[11] S. Casas, D. Cruz, G. Vidal, M. Constanzo, Uses and applications of the openapi/swagger specification: a systematic mapping of the literature, in: 2021 40th International Conference of the Chilean Computer Science Society (SCCC), IEEE, 2021, pp. 1–8.